REDMARS coin’s standardToken Smart Contract Audit Report - October 2021

Submitted By

Antier Solutions

Contents

Disclaimer

This is a limited audit report based on our analysis of the standardToken Smart Contract. It covers industry best practices as of the date of this report, concerning: smart contract best coding practices, cybersecurity vulnerabilities, issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks.

You are advised to read the full report to get a full view of our analysis. While we did our best in producing this report, it is important to note that you should not rely on this report, and cannot claim against us, based on what it says or does not say, or how we produced it, and you need to conduct your independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy all copies of this report downloaded and/or printed by you.

The report is provided "as is," without any condition, warranty, or other terms of any kind except as set out in this disclaimer. TAS hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might affect the report.

Executive Summary

REDMARS coin commissioned Antier Solutions to perform an end-to-end source code review of their Solidity Smart Contract. Team Antier Solutions (referred to as TAS throughout the report) performed the audit from October 18, 2021, to October 19, 2021.

The following report discusses severity issues and their scope of rectification through change recommendations. It also highlights activities that are successfully executed and others that need total reworking (if any).

The report emphasizes best practices in coding and the security vulnerabilities if any.

The information in this report should be used to understand the overall code quality, security, and correctness of the Smart Contract. The analysis is static and entirely limited to the Smart Contract code.

In the audit, we reviewed the Smart Contract’s code that implements the token mechanism.

Scope

We performed an independent technical audit to identify Smart Contract uncertainties. This shall protect the code from illegitimate authorization attempts or external & internal threats of any type. This also ensures end-to-end proofing of the contract from frauds.

The audit was performed semi-manually. We analyzed the Smart Contract code line-by-line and used an automation tool to report any suspicious code.

We considered the following standards for the Smart Contract code review:

- ERC20 [Token Contract best practices]

We used the following tools to perform automated tests:

- Ethereum Development platform:
  Hardhat
- Framework :
  Remix Ethereum
- Automation tools:
  - Slither
  - Surya
  - Mythril
  - Echidna

Audit Goals

The focus of the audit was to verify that the Smart Contract system was secure, resilient, and worked according to the specifications provided to the Auditing team.

TAS grouped the audit activities in the following three categories:

- **Security**
  Identifying security-related issues within each contract and the system of contracts

- **Architecture**
  Evaluation of the system architecture against smart contract conventions and general software best practices

- **Code Correctness and Quality**
  A full review of the contract source code. The primary areas of focus include:
  - Correctness
  - Readability
  - Sections of code with high complexity
  - Quantity and quality of test coverage

Result

The audit was conducted on the single contract file provided to the Auditing team. The following table provides an overall picture of the security posture. ✔ means no bugs were identified.

#	SMART CONTRACT PENETRATION TEST (OBJECTIVES)	AUDIT SUBCLASS	RESULT
1	OVERFLOW	-	✔
2	RACE CONDITION	-	✔
3	PERMISSIONS	-	✔
4	SAFETY DESIGN	OpenZeppelin safemath	✔
5	DDOS ATTACK	Call function security	✔
OVERALL SECURITY POSTURE		**Secure**	

Recommendations

The REDMARS coin's development team demonstrated high technical capabilities, both in the design of the token and implementation of the Smart Contract. Overall, the code includes effective use of abstraction, separation of concerns, and modularity.

Code security

TAS performed a static analysis of the code to identify possible loopholes. This verified whether the contract adhered to the Solidity best practices.

Implementation instructions

Include Unit Test cases for better understanding and testing of Gas limits. All the implementation instructions should be written in the README file.

Severity Level References

The following severity levels will describe the degree of every issue:

High severity issues

The issue puts the majority of, or large numbers of, users' sensitive information at risk, or is reasonably likely to lead to a catastrophic impact on the client's reputation or serious financial implications for the client and users.

Medium severity issues

The issue puts a subset of individual users' sensitive information at risk; exploitation would be detrimental to the client's reputation or is reasonably likely to lead to moderate financial impact.

Low severity issues

The risk is relatively low and could not be exploited regularly, or it's a risk not indicated as important or impactful by the client because of the client's business circumstances.

Informational

The issue does not pose an immediate threat to continued operation or usage but is relevant for security best practices, software engineering best practices, or defensive redundancy.

Optimization issue

The issue does not pose an immediate threat to continued operation or usage but is relevant for code optimization and gas efficiency best practices.

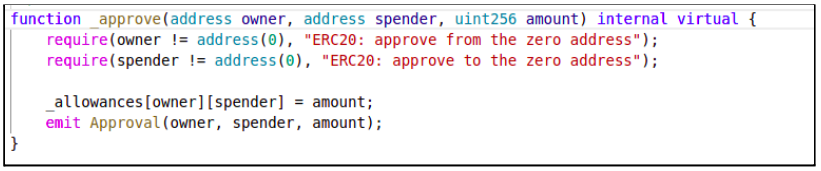
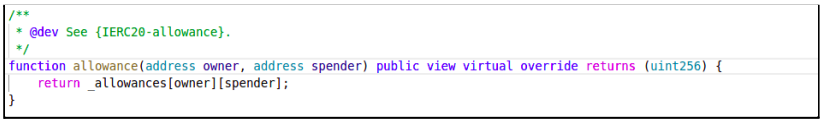
Number of vulnerabilities per severity

High	Medium	Low	Informational
0	1	3	4

Medium Severity Vulnerabilities

Severity	**Medium**
Contract	**standardToken**
Description	**Use Safe Math**
Code	You have imported SafeMath, but never used it.
Recommendation	Arithmetic operations in Solidity wrap on overflow. This can easily result in bugs because programmers usually assume that an overflow raises an error, which is the standard behavior in high-level programming languages. SafeMath restores this intuition by reverting the transaction when an operation overflows.
Status	

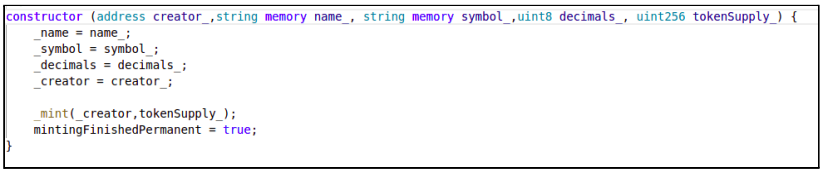
Low Severity Vulnerabilities

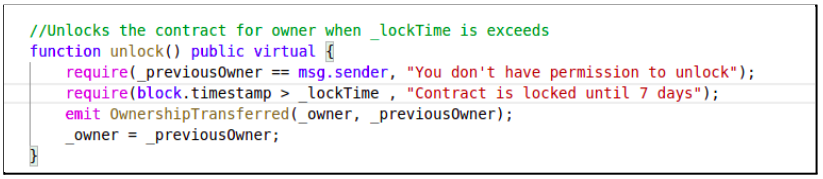
Severity	**Low**
Contract	**standardToken**
Description	**Detection of shadowing using local variables.**
Code	 
Recommendation	Rename the local variables that shadow another component.
Status	

```
function _approve(address owner, address spender, uint256 amount) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

```
/**
 * @dev See {IERC20-allowance}.
 */
function allowance(address owner, address spender) public view virtual override returns (uint256) {
    return _allowances[owner][spender];
}
```

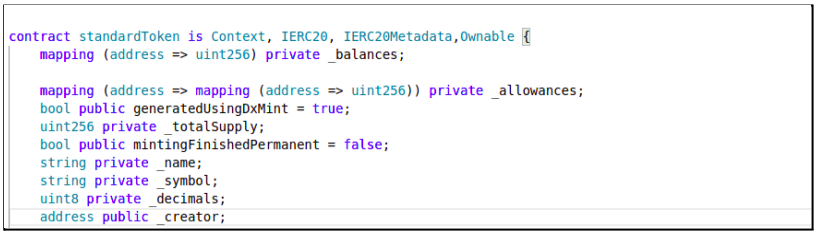
Severity	Low
Contract	standardToken
Description	Detects missing zero address validation.
Code	
Recommendation	Check that the address is not zero.
Status	

Severity	**Low**
Contract	**Ownable**
Description	**Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.**
Code	
Recommendation	Avoid relying on block.timestamp.
Status	

```
//Unlocks the contract for owner when _lockTime is exceeds
function unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock");
    require(block.timestamp > _lockTime , "Contract is locked until 7 days");
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
}
```

Informational Severity Vulnerabilities

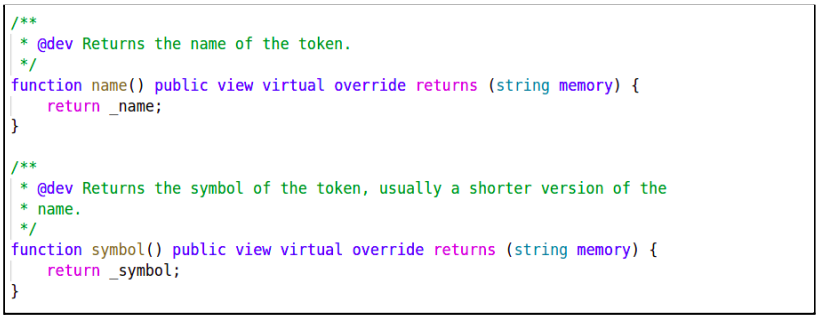
Severity	Informational
Contract	**standardToken**
Recommendation	**Deploy with any of the following Solidity versions:** • **0.5.16 - 0.5.17** • **0.6.11 - 0.6.12** • **0.7.5 - 0.7.6 Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.**
Code	`// SPDX-License-Identifier: MIT` `pragma solidity ^0.8.4;`
Description	solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statements.
Status	

Severity	Informational
Contract	**standardToken**
Recommendation	**Follow the Solidity naming convention.**
Code	
Description	Solidity defines a naming convention that should be followed.
Status	

```
contract standardToken is Context, IERC20, IERC20Metadata,Ownable {
    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;
    bool public generatedUsingDxMint = true;
    uint256 private _totalSupply;
    bool public mintingFinishedPermanent = false;
    string private _name;
    string private _symbol;
    uint8 private _decimals;
    address public _creator;
```

Severity	Informational
Contract	**standardToken**
Recommendation	**Remove the state variables that are never used.**
Code	`bool public generatedUsingDxMint = true;`
Description	Remove state variables that are never used to save gas.
Status	

Severity	Informational
Contract	**standardToken**
Recommendation	**Use the external attribute for functions never called from the contract.**
Code	
Description	public functions that are never called by the contract should be declared external to save gas.
Status	

```
/**
 * @dev Returns the name of the token.
 */
function name() public view virtual override returns (string memory) {
    return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view virtual override returns (string memory) {
    return _symbol;
}
```

Technical Analysis

The following is our automated and manual analysis of the standardToken Smart Contract code:

Checked Vulnerabilities

We checked standardToken Smart Contract for commonly known and specific business logic vulnerabilities. Following is the list of vulnerabilities tested in the Smart Contract code:

- Reentrancy
- Timestamp Dependence
- Gas limit and loops
- DoS with (unexpected) throw
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility

Automation Report

TAS used Slither, Mythril and Echidna to generate the automation report.

Slither

Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

```
INFO:Printers:
Compiled with Builder
Number of lines: 652 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 18
Number of informational issues: 15
Number of low issues: 4
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+---------------+-------------+-------+--------------------+--------------+----------+
|      Name     | # functions |  ERCS |     ERC20 info     | Complex code | Features |
+---------------+-------------+-------+--------------------+--------------+----------+
|    SafeMath   |      8      |       |                    |      No      |          |
| standardToken |      36     | ERC20 |     No Minting     |      No      |          |
|               |             |       | Approve Race Cond. |              |          |
|               |             |       |                    |              |          |
+---------------+-------------+-------+--------------------+--------------+----------+
INFO:Slither:. analyzed (6 contracts)
```

```
INFO:Detectors:
Redundant expression "this (contracts/standardToken.sol#124)" inContext (contracts/standardToken.sol#118-127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
standardToken.generatedUsingDxMint (contracts/standardToken.sol#380) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-
constant
```

```
INFO:Detectors:
standardToken.allowance(address,address).owner (contracts/standardToken.sol#466) shadows:
        - Ownable.owner() (contracts/standardToken.sol#318-320) (function)
standardToken._approve(address,address,uint256).owner (contracts/standardToken.sol#629) shadows:
        - Ownable.owner() (contracts/standardToken.sol#318-320) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
standardToken.constructor(address,string,string,uint8,uint256).creator_ (contracts/standardToken.sol#394) lacks
a zero-check on :
                - _creator = creator_ (contracts/standardToken.sol#398)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Ownable.unlock() (contracts/standardToken.sol#365-370) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > _lockTime,Contract is locked until 7 days) (contracts/standardT
oken.sol#367)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Context._msgData() (contracts/standardToken.sol#123-126) is never used and should be removed
SafeMath.add(uint256,uint256) (contracts/standardToken.sol#154-159) is never used and should be removed
SafeMath.div(uint256,uint256) (contracts/standardToken.sol#228-230) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/standardToken.sol#244-250) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/standardToken.sol#264-266) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/standardToken.sol#280-283) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/standardToken.sol#202-214) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/standardToken.sol#171-173) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (contracts/standardToken.sol#185-190) is never used and should be removed
standardToken._burn(address,uint256) (contracts/standardToken.sol#603-614) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (contracts/standardToken.sol#6) necessitates a version too recent to be trusted. Consider d
eploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Contract standardToken (contracts/standardToken.sol#376-653) is not in CapWords
Variable standardToken._creator (contracts/standardToken.sol#386) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventi
ons
```

```
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (contracts/standardToken.sol#318-320)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/standardToken.sol#337-340)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/standardToken.sol#346-350)
geUnlockTime() should be declared external:
        - Ownable.geUnlockTime() (contracts/standardToken.sol#352-354)
lock(uint256) should be declared external:
        - Ownable.lock(uint256) (contracts/standardToken.sol#357-362)
unlock() should be declared external:
        - Ownable.unlock() (contracts/standardToken.sol#365-370)
name() should be declared external:
        - standardToken.name() (contracts/standardToken.sol#407-409)
symbol() should be declared external:
        - standardToken.symbol() (contracts/standardToken.sol#415-417)
decimals() should be declared external:
        - standardToken.decimals() (contracts/standardToken.sol#432-434)
totalSupply() should be declared external:
        - standardToken.totalSupply() (contracts/standardToken.sol#439-441)
balanceOf(address) should be declared external:
        - standardToken.balanceOf(address) (contracts/standardToken.sol#446-448)
transfer(address,uint256) should be declared external:
        - standardToken.transfer(address,uint256) (contracts/standardToken.sol#458-461)
```

```
allowance(address,address) should be declared external:
        - standardToken.allowance(address,address) (contracts/standardToken.sol#466-468)
approve(address,uint256) should be declared external:
        - standardToken.approve(address,uint256) (contracts/standardToken.sol#477-480)
transferFrom(address,address,uint256) should be declared external:
        - standardToken.transferFrom(address,address,uint256) (contracts/standardToken.sol#495-503)
increaseAllowance(address,uint256) should be declared external:
        - standardToken.increaseAllowance(address,uint256) (contracts/standardToken.sol#517-520)
decreaseAllowance(address,uint256) should be declared external:
        - standardToken.decreaseAllowance(address,uint256) (contracts/standardToken.sol#536-542)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-
external
INFO:Slither:. analyzed (6 contracts with 75 detectors), 37 result(s) found
```

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: Ownable
Function name: unlock()
PC address: 426
Estimated Gas Usage: 1808 - 1903
A control flow decision is made based on The block.timestamp environment variable.
The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timest
amp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment varia
bles as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
--------------------
In file: standardToken.sol:367
```

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: Ownable
Function name: lock(uint256)
PC address: 1063
Estimated Gas Usage: 12834 - 52929
A control flow decision is made based on The block.timestamp environment variable.
The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timest
amp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment varia
bles as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
--------------------
In file: #utility.yul:55
```

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron, and other EVM-compatible blockchains. It uses symbolic execution, SMT solving, and taint analysis to detect a variety of security vulnerabilities.

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: Ownable
Function name: unlock()
PC address: 426
Estimated Gas Usage: 1808 - 1903
A control flow decision is made based on The block.timestamp environment variable.
The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timest
amp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment varia
bles as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
--------------------
In file: standardToken.sol:367
```

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: Ownable
Function name: lock(uint256)
PC address: 1063
Estimated Gas Usage: 12834 - 52929
A control flow decision is made based on The block.timestamp environment variable.
The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timest
amp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment varia
bles as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
--------------------
In file: #utility.yul:55
```

Echidna

Echidna is a weird creature that eats bugs and is highly electrosensitive. More seriously, Echidna is a Haskell program designed for fuzzing/property-based testing of smart contracts. It uses sophisticated grammar-based fuzzing campaigns based on a contract ABI to falsify user-defined predicates or Solidity assertions. We designed Echidna with modularity in mind, so it can be easily extended to include new mutations or test specific contracts in specific cases.

```
Analyzing contract: /home/surya/Documents/Audit/standardToken/contracts/TeststandardTokenTransferable.sol:Teststandar
dTokenTransferable
crytic_totalSupply_consistant_ERC20Properties: passed! 🎉
crytic_approve_overwrites: passed! 🎉
crytic_self_transferFrom_to_other_ERC20PropertiesTransferable: passed! 🎉
crytic_zero_always_empty_ERC20Properties: passed! 🎉
crytic_self_transfer_ERC20PropertiesTransferable: passed! 🎉
crytic_self_transferFrom_ERC20PropertiesTransferable: passed! 🎉
crytic_revert_transferFrom_to_zero_ERC20PropertiesTransferable: passed! 🎉
crytic_revert_transfer_to_user_ERC20PropertiesTransferable: passed! 🎉
crytic_revert_transfer_to_zero_ERC20PropertiesTransferable: passed! 🎉
crytic_transfer_to_other_ERC20PropertiesTransferable: passed! 🎉
crytic_less_than_total_ERC20Properties: passed! 🎉

Unique instructions: 2960
Unique codehashes: 1
Seed: -7618994799742055016
```

Functional test

Function Names	Testing results
name	Pass
symbol	Pass
decimals	Pass
totalSupply	Pass
balanceOf	Pass
transfer	Pass
allowance	Pass
approve	Pass
transferFrom	Pass
increaseAllowance	Pass
decreaseAllowance	Pass
_transfer	Pass
_mint	Pass
_burn	Pass
_approve	Pass
_beforeTokenTransfer	Pass

Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of standardToken and methods for exploiting them. Antier Solutions recommends that precautions should be taken to protect the confidentiality of this document and the information contained herein.

Security Assessment is an uncertain process based on experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, although Antier Solutions has identified major security vulnerabilities of the analyzed system, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures.

As technologies and risks change over time, the vulnerabilities associated with the operation of the standardToken Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Antier Solutions makes no undertaking to supplement or update this report based on the changed circumstances or facts of which Antier Solutions becomes aware after the date hereof.

This report may recommend that Antier Solutions use certain software or hardware products manufactured or maintained by other vendors. Antier Solutions bases these recommendations on its prior experience with the capabilities of those products. Nonetheless, Antier Solutions does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended.

The Non-Disclosure Agreement (NDA) in effect between Antier Solutions and REDMARS Coin governs the disclosure of this report to all other parties, including product vendors and suppliers.